

## Tema 2: Búsqueda y programación lógica

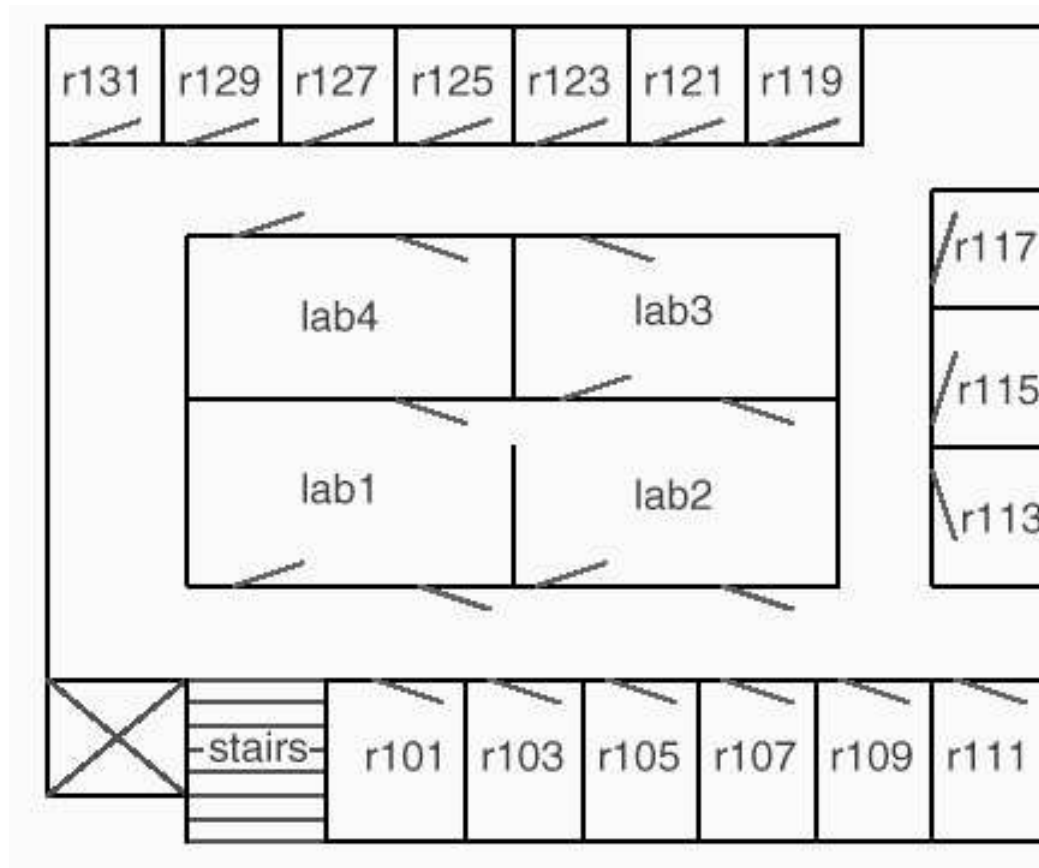
José A. Alonso Jiménez  
Miguel A. Gutiérrez Naranjo  
Francisco J. Martín Mateos

Dpto. de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE SEVILLA

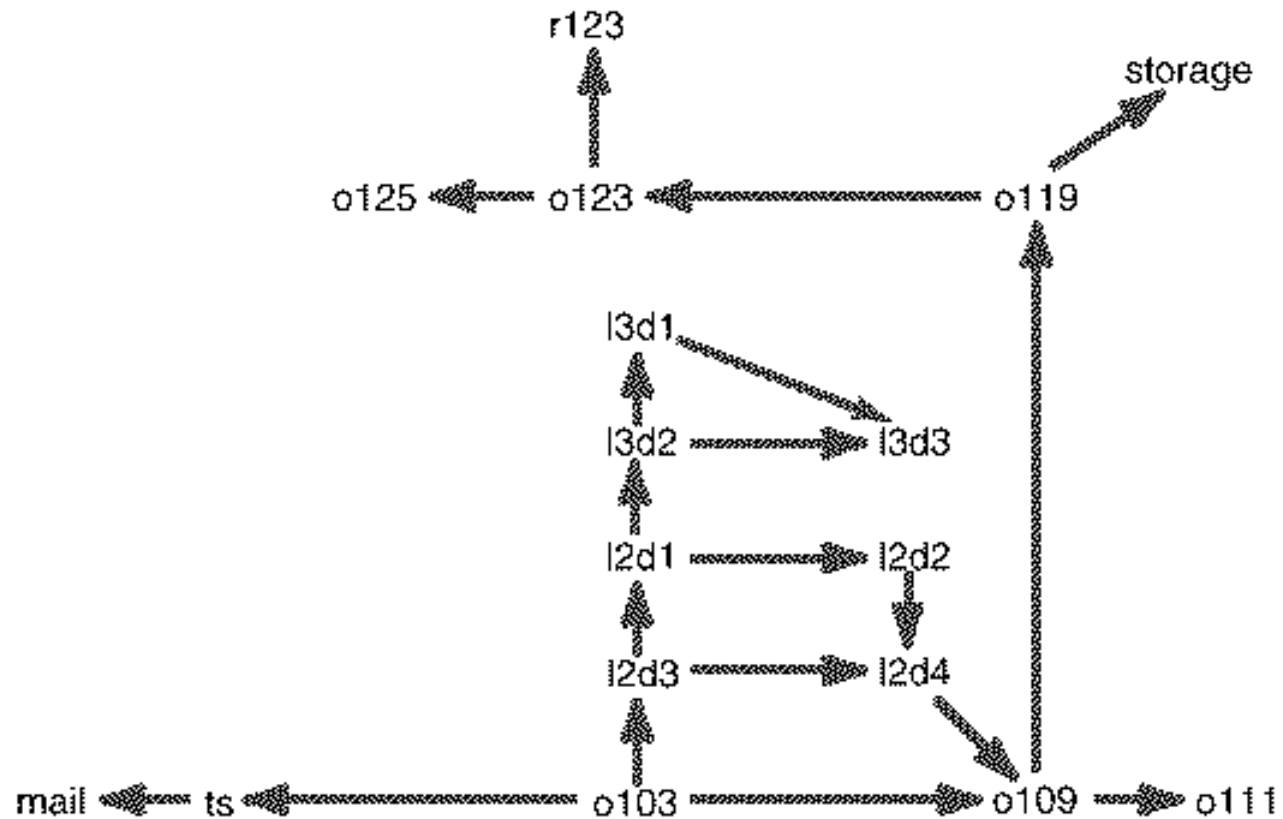
# El robot repartidor

El mundo del robot repartidor (Poole-98 p. 14)



# El robot repartidor

El mundo del robot repartidor (Poole-98 p. 14)



# El robot repartidor

- **Lenguaje**
  - f103 en frente la habitación 103
  - fe en frente la escalera
  - correo en el correo
  - l2p3 en la puerta 3 del laboratorio 2
  - almacén en el almacén
  - h123 en la habitación 123
- **Estado inicial**  
`estado_inicial(f103).`
- **Estados finales**  
`estado_final(h123).`

# El robot repartidor

- **Sucesores**

```
sucesor(f103,fe).      sucesor(f103,l2p3).      sucesor(f103,f109).
sucesor(fe,correo).   sucesor(f109,f111).     sucesor(f109,f119).
sucesor(f119,almacén).sucesor(f119,f123).     sucesor(f123,h123).
sucesor(f123,f125).   sucesor(l2p1,l3p2).     sucesor(l2p1,l2p2).
sucesor(l2p2,l2p4).   sucesor(l2p3,l2p1).     sucesor(l2p3,l2p4).
sucesor(l2p4,f109).   sucesor(l3p2,l3p3).     sucesor(l3p2,l3p1).
sucesor(l3p1,l3p3).
```

- **Coste**

```
coste(E1,E2,C) :-
    posicion(E1,X1,Y1),
    posicion(E2,X2,Y2),
    C is abs(X1-X2)+abs(Y1-Y2).
```

# El robot repartidor

```
posicion(correo,17,43).  posicion(fe,23,43).  posicion(f103,31,43).  
posicion(f109,43,43).  posicion(f111,47,43).  posicion(f119,42,58).  
posicion(f123,33,58).  posicion(f125,29,58).  posicion(h123,33,62).  
posicion(l2p1,33,49).  posicion(l2p2,39,49).  posicion(l2p3,32,46).  
posicion(l2p4,39,46).  posicion(l3p1,34,55).  posicion(l3p2,33,52).  
posicion(l3p3,39,52).  posicion(almacén,45,62).
```

- **Heurística**

```
heuristica(E,H) :-  
    posicion(E,X,Y),  
    estado_final(E1),  
    posicion(E1,X1,Y1),  
    H is abs(X-X1)+abs(Y-Y1).
```

# Procedimiento general de búsqueda

- Relaciones dependientes del problema
  - $\text{estado\_inicial}(E)$  se verifica si  $E$  es el estado inicial
  - $\text{estado\_final}(E)$  se verifica si  $E$  es un estado final
  - $\text{sucesor}(E_1, E_2)$  se verifica si  $E_2$  es un estado sucesor de  $E_1$
  - $\text{coste}(E_1, E_2, C)$  se verifica si  $C$  es el coste de ir del estado  $E_1$  al  $E_2$
  - $\text{heuristica}(E, H)$  que se verifica si  $H$  es la heurística del estado  $E$
- Datos:
  - Un *nodo* es una lista de estados  $[E_n, \dots, E_1]$  de forma que  $E_1$  es el estado inicial y  $E_{(i+1)}$  es un sucesor de  $E_i$
  - *Abiertos* es una lista de nodos (los nodos pendientes de analizar)

# Procedimiento general de búsqueda

- **Procedimiento general de búsqueda**

- `busqueda(+M,?S)` se verifica si `S` es una solución del problema mediante búsqueda según el método `M`

- **Procedimiento:**

1. Sea `E` el estado inicial.

2. La solución `S` es la obtenida por búsqueda según el método `M` con `[[E]]` como la lista de abiertos.

- **Definición**

```
busqueda(M,S) :-  
    estado_inicial(E),      % 1  
    busqueda(M,[[E]],S).   % 2
```



# Procedimiento general de búsqueda

- **Procedimiento auxiliar de búsqueda**

- `busqueda(+M,+Abiertos,?S)` se verifica si `S` es una solución encontrada por búsqueda según el método `M` a partir de las listas de `Abiertos`

- **Procedimiento:**

1. Si 1.1. el primer elemento de `Abiertos` es `[E|C]` y

- 1.2. `E` es un estado final,

entonces

- 1.3 `S` es la inversa de `[E|C]`.

2. Si 2.1. `N` es un nodo de `Abiertos` (seleccionado según el método `M`) y `R` son los restantes nodos de `Abiertos`,

- 2.2. `Sucesores` es la lista de los sucesores del nodo `N`,

- 2.3. los nuevos abiertos, `NAbiertos`, es la lista obtenida expandiendo (según el método `M`) `R` con los `Sucesores`

entonces

- 2.4. `S` es la solución obtenida por búsqueda (según el método `M`) con los nuevos abiertos.

# Procedimiento general de búsqueda

- Definición

```
busqueda(_M,Abiertos,S) :-  
    Abiertos = [[E|C]|_],           % 1.1  
    estado_final(E),                % 1.2  
    reverse([E|C],S).               % 1.3  
busqueda(M,Abiertos,S) :-  
    selecciona(M,Abiertos,N,R),      % 2.1  
    % write(N), nl,  
    sucesores(N,Sucesores),         % 2.2  
    expande(M,R,Sucesores,NAbiertos), % 2.3  
    busqueda(M,NAbiertos,S).        % 2.4
```

# Procedimiento general de búsqueda

- `selecciona(+M,+LN1,?N,?LN2)` se verifica si `N` es un nodo de la lista `LN1` y `LN2` es la lista de los restantes nodos.  
:- `discontiguous selecciona/4`.
- `sucesores(+N,?L)` se verifica si `L` es la lista de los sucesores del nodo `N`  
`sucesores([E|C],L) :-`  
    `findall([E1,E|C],sucesor(E,E1),L).`
- `expande(+M,+L1,+Sucesores,?L2)` se verifica si `L2` es la lista expandiendo (según el método `M`) la lista de nodos `L1` con la lista de nodos `Sucesores`  
:- `discontiguous expande/4`.

# Procedimiento general de búsqueda

- Búsqueda en anchura

- Definición

`selecciona(anchura, [N|R], N, R) .`

`expande(anchura, L1, Sucesores, L2) :-  
append(L1, Sucesores, L2) .`

# Procedimiento general de búsqueda

```
?- busqueda(anchura,S).  
[f103]  
[fe, f103]  
[l2p3, f103]  
[f109, f103]  
[correo, fe, f103]  
[l2p1, l2p3, f103]  
[l2p4, l2p3, f103]  
[f111, f109, f103]  
[f119, f109, f103]  
[l3p2, l2p1, l2p3, f103]  
[l2p2, l2p1, l2p3, f103]  
[f109, l2p4, l2p3, f103]  
[almacén, f119, f109, f103]  
[f123, f119, f109, f103]  
[l3p3, l3p2, l2p1, l2p3, f103]  
[l3p1, l3p2, l2p1, l2p3, f103]  
[l2p4, l2p2, l2p1, l2p3, f103]  
[f111, f109, l2p4, l2p3, f103]  
[f119, f109, l2p4, l2p3, f103]  
S = [f103, f109, f119, f123, h123] ;
```

## Procedimiento general de búsqueda

[h123, f123, f119, f109, f103]  
[f125, f123, f119, f109, f103]  
[l3p3, l3p1, l3p2, l2p1, l2p3, f103]  
[f109, l2p4, l2p2, l2p1, l2p3, f103]  
[almacén, f119, f109, l2p4, l2p3, f103]  
[f123, f119, f109, l2p4, l2p3, f103]  
[f111, f109, l2p4, l2p2, l2p1, l2p3, f103]  
[f119, f109, l2p4, l2p2, l2p1, l2p3, f103]  
S = [f103, l2p3, l2p4, f109, f119, f123, h123]

Yes

# Procedimiento general de búsqueda

- **Búsqueda en profundidad**

```
selecciona(profundidad, [N|R], N, R).
```

```
expande(profundidad, L1, Sucesores, L2) :-  
    append(Sucesores, L1, L2).
```

- **Búsqueda optimal**

- **Definición**

```
expande(optimal, L1, Sucesores, L2) :-  
    append(Sucesores, L1, L2).
```

```
selecciona(optimal, LN1, N, LN2) :-  
    selecciona_con_valor(optimal, LN1, N, LN2).
```

# Procedimiento general de búsqueda

- `selecciona_con_valor(+M,+LN1,?N,?LN2)` se verifica si `N` es el mejor nodo (según el método `M`) de la lista `LN1` y `LN2` es la lista de los restantes nodos.

```
selecciona_con_valor(M, LN1, N, LN2) :-  
    member(N, LN1),  
    valor(M, N, V),  
    not(member(N1, LN1),  
         valor(M, N1, V1),  
         V1 < V),  
    select(N.LN1, LN2).
```

- `valor(+M,+N,?V)` se verifica si el valor (según el método `M`) del nodo `N` es `V`

```
:- disjointous valor/3.
```

```
valor(optimal, N, V) :-  
    coste_camino(N, V).
```



# Procedimiento general de búsqueda

- `coste_camino(+N,?V)` se verifica si  $V$  es el coste del camino representado por el nodo  $N$

```
coste_camino([_E],0).
coste_camino([E2,E1|R],V) :-
    coste(E2,E1,V1),
    coste_camino([E1|R],V2),
    V is V1+V2.
```

- **Búsqueda por primero el mejor**

```
selecciona(primero_el_mejor,LN1,N,LN2) :-
    selecciona_con_valor(primero_el_mejor,LN1,N,LN2).
```

```
valor(primero_el_mejor,[E|R],V) :-
    heuristica(E,V).
```

```
expande(primero_el_mejor,L1,Sucesores,L2) :-
    append(Sucesores,L1,L2).
```

# Procedimiento general de búsqueda

- Búsqueda por A\*

```
selecciona(a_estrella, LN1, N, LN2) :-  
    selecciona_con_valor(a_estrella, LN1, N, LN2).
```

```
valor(a_estrella, [E|R], V) :-  
    coste_camino([E|R], V1),  
    heuristica(E, V2),  
    V is V1+V2.
```

```
expande(a_estrella, L1, Sucesores, L2) :-  
    append(Sucesores, L1, L2).
```

# Procedimiento general de búsqueda sin reevaluaciones

- Datos:

- Un *nodo* es un término  $V-[E_n, \dots, E_1]$  de forma que  $E_1$  es el estado inicial,  $E_{(i+1)}$  es un sucesor de  $E_i$  y  $V$  es el valor de  $[E_n, \dots, E_1]$  según el procedimiento de búsqueda.
- *Abiertos* es una lista de nodos (los nodos pendientes de analizar).

- Procedimiento general de búsqueda

```
busqueda(M,S) :-  
    estado_inicial(E),  
    valor(M,0-[],E,V),  
    busqueda(M,[V-[E]],S).
```

# Procedimiento general de búsqueda sin reevaluaciones

```
busqueda(_M,Abiertos,S) :-  
    Abiertos = [_-[E|C]|_],  
    estado_final(E),  
    reverse([E|C],S).  
busqueda(M,Abiertos,S) :-  
    selecciona(M,Abiertos,N,R),  
    sucesores(M,N,Sucesores),  
    expande(M,R,Sucesores,NAbiertos),  
    busqueda(M,NAbiertos,S).  
  
selecciona(_M,[N|R],N,R).  
  
sucesores(M,V-[E|C],L) :-  
    findall(V1-[E1,E|C],  
        (sucesor(E,E1),valor(M,V-[E|C],E1,V1)),  
        L).  
  
expande(_M,L1,Sucesores,L2) :-  
    append(Sucesores,L1,L3), sort(L3,L2).
```

# Procedimiento general de búsqueda sin reevaluaciones

- **Búsqueda optimal**

```
valor(optimal,0-[],_E,0).  
valor(optimal,V-[E|_C],E1,V1) :- coste(E,E1,V2), V1 is V+V2.
```

- **Búsqueda por primero el mejor**

```
valor(primer_el_mejor,_N,E,V) :- heuristica(E,V).
```

- **Búsqueda por A\***

```
valor(a_estrella,0-[],E,H+0) :- heuristica(E,H).  
valor(a_estrella,_F+C-[E|_R],E1,F1+C1) :-  
    coste(E,E1,C2),  
    C1 is C+C2,  
    heuristica(E1,H),  
    F1 is C1+H.
```

# Procedimiento general de búsqueda sin reevaluaciones

- Sesión

```
?- busqueda(a_estrella,S).  
21+0-[f103]  
21+4-[l2p3, f103]  
21+8-[l2p1, l2p3, f103]  
21+11-[l3p2, l2p1, l2p3, f103]  
23+15-[l3p1, l3p2, l2p1, l2p3, f103]  
33+11-[l2p4, l2p3, f103]  
33+14-[l2p2, l2p1, l2p3, f103]  
33+17-[l3p3, l3p2, l2p1, l2p3, f103]  
37+8-[fe, f103]  
39+17-[l2p4, l2p2, l2p1, l2p3, f103]  
39+23-[l3p3, l3p1, l3p2, l2p1, l2p3, f103]  
41+12-[f109, f103]  
41+28-[f119, f109, f103]  
41+37-[f123, f119, f109, f103]  
S = [f103, f109, f119, f123, h123]  
Yes
```

## Refinamientos de estrategias de búsqueda

- Eliminación de ciclos
- Eliminación de caminos múltiples
- Búsqueda por profundidad acotada
- Búsqueda en profundidad iterativa
- Búsqueda en haz
- Búsqueda en escalada

## Bibliografía

- Flach, P. *Simply Logical (Intelligent Reasoning by Example)* (John Wiley, 1994)
  - Cap. 5: “Seaching graphs”
  - Cap. 6: “Informed search”
- Poole, D.; Mackworth, A. y Goebel, R. *Computational Intelligence (A Logical Approach)* (Oxford University Press, 1998)
  - Cap. 4: “Searching”
- Shoham, Y. *Artificial Intelligence Techniques in Prolog* (Morgan Kaufmann, 1994)
  - Cap. 2 “Search”